

This Page Is Inserted by IFW Operations  
and is not a part of the Official Record

## **BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning documents *will not* correct images,  
please do not report the images to the  
Image Problem Mailbox.**

#3  
THE COMMISSIONER IS AUTHORIZED  
TO CHARGE ANY DEFICIENCY IN THE  
FEES FOR THIS PAPER TO DEPOSIT  
ACCOUNT NO. 23-0975

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of

Shinichiro ETO et al.

Serial No. NEW

Filed April 19, 2001

REAL-TIME OS SIMULATOR

: Attn: APPLICATION BRANCH

: Attorney Docket No. 2001\_0469A

1000 U.S. PRO  
09/837205  
04/19/01

CLAIM OF PRIORITY UNDER 35 USC 119

Assistant Commissioner for Patents,  
Washington, DC 20231

Sir:

Applicants in the above-entitled application hereby claim the date of priority under the International Convention of Japanese Patent Application No. 2000-120903, filed April 21, 2000, as acknowledged in the Declaration of this application.

A certified copy of said Japanese Patent Application is submitted herewith.

Respectfully submitted,

Shinichiro ETO et al.

By Charles R. Watts  
Charles R. Watts  
Registration No. 33,142  
Attorney for Applicants

CRW/asd  
Washington, D.C. 20006-1021  
Telephone (202) 721-8200  
Facsimile (202) 721-8250  
April 19, 2001

日 本 国 特 許 庁  
PATENT OFFICE  
JAPANESE GOVERNMENT

J1000 U.S. PTO  
09/837205  
04/19/01

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日  
Date of Application:

2000年 4月21日

出 願 番 号  
Application Number:

特願2000-120903

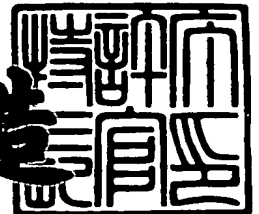
出 願 人  
Applicant(s):

松下電器産業株式会社

2000年12月15日

特許庁長官  
Commissioner,  
Patent Office

及川耕造



出証番号 出証特2000-3104158

【書類名】 特許願

【整理番号】 2022520081

【あて先】 特許庁長官殿

【国際特許分類】 G06F 9/46

【発明者】

    【住所又は居所】 広島県東広島市鏡山 3 丁目 1 0 番 1 8 号 株式会社松下  
                          電器情報システム広島研究所内

    【氏名】 衛藤 眞一郎

【発明者】

    【住所又は居所】 広島県東広島市鏡山 3 丁目 1 0 番 1 8 号 株式会社松下  
                          電器情報システム広島研究所内

    【氏名】 枯木 正吉

【特許出願人】

    【識別番号】 000005821

    【氏名又は名称】 松下電器産業株式会社

【代理人】

    【識別番号】 100098291

    【弁理士】

    【氏名又は名称】 小笠原 史朗

【手数料の表示】

    【予納台帳番号】 035367

    【納付金額】 21,000円

【提出物件の目録】

    【物件名】 明細書 1

    【物件名】 図面 1

    【物件名】 要約書 1

    【包括委任状番号】 9405386

【プルーフの要否】 要

【書類名】 明細書

【発明の名称】 リアルタイムOSシミュレータ

【特許請求の範囲】

【請求項1】 リアルタイムOS上で実行される複数のタスクのそれぞれに汎用のマルチスレッドOS上で実行されるタスク処理スレッドを割り当て、前記リアルタイムOSの動作を前記マルチスレッドOS上で模倣するリアルタイムOSシミュレータであって、

前記リアルタイムOSと同一の条件下で前記タスク処理スレッドが発する要求を受け付け、前記要求に応じてタスクの切り替えを指示するタスク切り替え指示手段と、

前記タスク切り替え指示手段と協働して、前記マルチスレッドOSの機能を用いて前記タスク処理スレッドを停止および再開させることにより、前記タスク処理スレッドの中から選択した一のタスク処理スレッドを実行させるタスク切り替えスレッドとを備えた、リアルタイムOSシミュレータ。

【請求項2】 前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択し、前記タスク切り替えスレッドに対して処理開始を指示した後に、前記要求を発したタスク処理スレッドを停止させ、

前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行されていたタスク処理スレッドが停止した後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする、請求項1に記載のリアルタイムOSシミュレータ。

【請求項3】 前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行されていたタスク処理スレッドが停止したか否かを所定の時間間隔で調べることを特徴とする、請求項2に記載のリアルタイムOSシミュレータ。

【請求項4】 前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択し、前記タスク切り替えスレッドに対して処理開始を指示した後に、前記要求を発したタスク処理スレッドを待機状態にさせ、

前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行さ

れていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドの待機状態を解除するとともに、当該タスク処理スレッドを再開させることを特徴とする、請求項1に記載のリアルタイムOSシミュレータ。

【請求項5】 前記タスク切り替え指示手段は、前記タスク切り替えスレッドが処理可能状態となった後に、前記タスク切り替えスレッドに処理開始を指示することを特徴とする、請求項2または4に記載のリアルタイムOSシミュレータ。

【請求項6】 前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択した後に、前記タスク切り替えスレッドに対して処理開始を指示し、

前記タスク切り替えスレッドは、前記タスク処理スレッドよりも高い優先度で実行され、処理開始を指示された時には、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする、請求項1に記載のリアルタイムOSシミュレータ。

【請求項7】 前記タスク処理スレッドを生成するタスク処理スレッド生成手段をさらに備えた、請求項1に記載のリアルタイムOSシミュレータ。

【請求項8】 擬似的に割り込みを発生させる割り込みスレッドが発する割り込み要求を受け付け、実行中のタスク処理スレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、次に実行すべきタスク処理スレッドを選択して再開させる割り込み処理手段をさらに備えた、請求項1に記載のリアルタイムOSシミュレータ。

【請求項9】 前記割り込み処理手段は、前記割り込みスレッドから割り込み要求を受け付けた時に他の割り込みスレッドが実行されていた場合には、実行中の割り込みスレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、停止させた割り込みスレッドを再開させることを特徴とする、請求項8に記載のリアルタイムOSシミュレータ。

【請求項10】 前記割り込みスレッドの中に、所定の時間間隔で擬似的に割り込みを発生させるシステムクロック割り込みスレッドが含まれることを特徴とする、請求項8に記載のリアルタイムOSシミュレータ。

【請求項 1 1】 前記割り込みスレッドを生成する割り込みスレッド生成手段をさらに備えた、請求項 8 に記載のリアルタイム OS シミュレータ。

【請求項 1 2】 リアルタイム OS 上で実行される複数のタスクのそれぞれに汎用のマルチスレッド OS 上で実行されるタスク処理スレッドを割り当て、前記リアルタイム OS の動作を前記マルチスレッド OS 上で模倣するリアルタイム OS のシミュレーション方法をコンピュータで実行させるための記録媒体を記録したコンピュータ読み取り可能な記録媒体であって、

前記リアルタイム OS と同一の条件下で前記タスク処理スレッドが発する要求を受け付け、前記要求に応じてタスクの切り替えを指示する第 1 のステップと、

前記マルチスレッド OS の機能を用いて前記タスク処理スレッドを停止および再開させることにより、前記タスク処理スレッドの中から選択した一のタスク処理スレッドを実行させる第 2 のステップとを備えた、リアルタイム OS のシミュレーション方法をコンピュータに実行させるためのプログラムを記録したコンピュータ読み取り可能な記録媒体。

【請求項 1 3】 前記第 1 のステップは、次に実行すべきタスク処理スレッドを選択した後に、前記要求を発したタスク処理スレッドを停止させ、

前記第 2 のステップは、直前に実行されていたタスク処理スレッドが停止した後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする、請求項 1 2 に記載の記録媒体。

【請求項 1 4】 前記第 1 のステップは、次に実行すべきタスク処理スレッドを選択した後に、前記要求を発したタスク処理スレッドを待機状態にさせ、

前記第 2 のステップは、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドの待機状態を解除するとともに、当該タスク処理スレッドを再開させることを特徴とする、請求項 1 2 に記載の記録媒体。

【請求項 1 5】 前記第 1 のステップは、次に実行すべきタスク処理スレッドを選択し、

前記第 2 のステップは、前記タスク処理スレッドよりも高い優先度で実行され、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく

選択されたタスク処理スレッドを再開させることを特徴とする、請求項 1 2 に記載の記録媒体。

【請求項 1 6】 前記リアルタイム OS のシミュレーション方法は、擬似的に割り込みを発生させる割り込みスレッドが発する割り込み要求を受け付け、実行中のタスク処理スレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、次に実行すべきタスク処理スレッドを選択して再開させる第 3 のステップをさらに備えた、請求項 1 2 に記載の記録媒体。

【発明の詳細な説明】

【0 0 0 1】

【発明の属する技術分野】

本発明は、組み込み機器に内蔵されるアプリケーションソフトウェアを開発するために使用されるリアルタイム OS シミュレータに関する。

【0 0 0 2】

【従来の技術】

携帯電話やデジタル TV などの組み込み機器は、機器ごとに固有のアプリケーションソフトウェアを内蔵している。組み込み機器用のソフトウェアは、リアルタイム OS 上で実行されるマルチタスクのソフトウェアとして実現され、ブレッドボードや ICE (In Circuit Emulator) などからなる開発環境（以下、実システムという）を用いて開発される。しかし、実システムを用いる方法では、実システム完成後のソフトウェア開発期間が短くなり、実システムがソフトウェア開発者に比べて不足するといった問題が生じる。また、実システムの動作周波数上昇に伴い、ICE を用いたデバッグ自体が困難になるという問題も生じている。

【0 0 0 3】

この問題を解決する 1 つの手法として、汎用のパソコンやワークステーション上に、実システムと同様の動作を行うソフトウェア開発システムを構成し、これを用いて組み込み機器用のソフトウェアを開発する方法が考えられる。例えば、実システムにおけるタスクを汎用のマルチスレッド OS（以下、汎用 OS という）上のスレッドと 1 対 1 に対応づけ、リアルタイム OS の動作を模倣する「リア

ルタイムOSシミュレータ」を用いる方法がある。この方法によれば、汎用OSが提供するマルチスレッド機能を用いてリアルタイムOSのマルチタスク機能を模倣すればよいので、リアルタイムOSシミュレータを容易に実現することができる。また、リアルタイムOSシミュレータのみを開発すれば、十分な数のソフトウェア開発システムをソフトウェア開発者に提供し、ソフトウェアの開発効率と信頼性とを向上させることができる。なお、以下でスレッドという時には、一部の汎用OSにおいてプロセスと呼ばれるものを含むものとする。

## 【0004】

リアルタイムOSは、タスク同期管理、タイマー管理、タスク間通信、および、メモリ管理などの様々な機能を有する。リアルタイムOSは、このうち、実行中のタスクを切り替えるディスパッチ処理、および、発生した割り込みに対して所定の割り込みハンドラを呼び出す割り込み処理を実現するにあたり、実システムのハードウェアを用いる。これに対し、リアルタイムOSシミュレータは、実システムのハードウェアを用いることができないため、この2つの処理を行うことが難しい。

## 【0005】

リアルタイムOSは、割り込みハンドラを実行中に他の割り込みが発生した場合に、後に発生した割り込みを先に処理する多重割り込み処理を行う場合がある。また、リアルタイムOSは、タスクまたは割り込みハンドラ内で呼び出された時に、必要に応じてディスパッチ処理を行う。タスク内でリアルタイムOSが呼び出された時には、その時点でディスパッチ処理が行われる。これに対し、割り込みハンドラ内でリアルタイムOSが呼び出された時には、割り込みハンドラが終了した時点でディスパッチ処理（以下、遅延ディスパッチ処理という）が行われる。リアルタイムOSシミュレータは、このような遅延ディスパッチ処理や多重割り込み処理を含めて、リアルタイムOSの機能を汎用OS上で忠実に模倣する必要がある。

## 【0006】

図18は、日本国特許第2820189号によって開示された、数値制御装置の制御ソフトウェア実行システムのソフトウェア構成図である。なお、図18は

、本願発明との対比を容易にするために、発明の本質を損なわない程度に抽象化されている。図 1 8 に示す 4 つのスレッドは、汎用 OS を搭載したホスト計算機上で並行に実行される。3 つのスレッド 9 1 ~ 9 3 は、このホスト計算機と異なる数値制御装置において実行されるタスクと同じ処理を行う。スケジューラスレッド 9 0 は、スレッド 9 1 ~ 9 3 に対して定期的に割り込み、スレッドの実行を能動的に切り替える。このソフトウェア実行システムによれば、数値制御装置の制御ソフトウェアをホスト計算機上で動作させることができる。

## 【 0 0 0 7 】

## 【発明が解決しようとする課題】

しかしながら、図 1 8 に示すスケジューラスレッド 9 0 は、以下に示すように、組み込み機器用のソフトウェア開発において使用されるリアルタイム OS シミュレータに必要な機能を果たすことができない。

## 【 0 0 0 8 】

まず、スケジューラスレッド 9 0 は、スレッド 9 1 ~ 9 3 に対して定期的に割り込み、実行スレッドを能動的に切り替えるプリエンプティブなスレッド制御を行う。これに対して、実システムでは、タスクがリアルタイム OS を呼び出した時に実行タスクを切り替えるノンプリエンプティブなタスク制御が行われる。従来の方法では、リアルタイム OS の特徴であるノンプリエンプティブなタスク制御を行うことができない。

## 【 0 0 0 9 】

次に、スケジューラスレッド 9 0 は、汎用 OS がすべてのスレッドを平等に制御するとの仮定のもとで、スレッド 9 1 ~ 9 3 に定期的に割り込み、実行スレッドを切り替える。しかし、一般に汎用 OS のスレッドスケジューリングアルゴリズムは公開されない。また、一部の汎用 OS は、指定されたスレッド優先度に基づき内部で別途スレッド優先度を算出し、算出したスレッド優先度に従ってスレッドの実行を制御する。このような汎用 OS ではアプリケーションソフトウェアでスレッド優先度を設定するだけでは、スレッドの実行を制御することができない。したがって、このような汎用 OS を用いて従来の方法を実施する場合には、スレッド 9 1 ~ 9 3 のみが実行され、スケジューラスレッド 9 0 が全く実行され

ない可能性がある。

【 0 0 1 0 】

さらに、スケジューラスレッド 9 0 は、リアルタイム OS シミュレータに必要なとされる割り込み処理機能を有していない。組み込み機器用のソフトウェアは、外部からの割り込みに応答して動作する。しかし、従来の方法では、このような割り込み処理を扱うことができない。

【 0 0 1 1 】

それ故に、本発明は、任意のスレッドスケジューリングアルゴリズムを用いた汎用のマルチスレッド OS 上においても、リアルタイム OS のディスパッチ処理と割り込み処理とを忠実に模倣するリアルタイム OS シミュレータを提供することを目的とする。

【 0 0 1 2 】

【課題を解決するための手段および発明の効果】

第 1 の発明は、リアルタイム OS 上で実行される複数のタスクのそれぞれに汎用のマルチスレッド OS 上で実行されるタスク処理スレッドを割り当て、前記リアルタイム OS の動作を前記マルチスレッド OS 上で模倣するリアルタイム OS シミュレータであって、

前記リアルタイム OS と同一の条件下で前記タスク処理スレッドが発する要求を受け付け、前記要求に応じてタスクの切り替えを指示するタスク切り替え指示手段と、

前記タスク切り替え指示手段と協働して、前記マルチスレッド OS の機能を用いて前記タスク処理スレッドを停止および再開させることにより、前記タスク処理スレッドの中から選択した一のタスク処理スレッドを実行させるタスク切り替えスレッドとを備える。

【 0 0 1 3 】

このような第 1 の発明によれば、タスク切り替え指示手段とタスク切り替えスレッドとの作用により、一のタスク処理スレッドのみが切り替えて実行される。これにより、マルチスレッド OS が提供するスレッドスケジューリングアルゴリズムにかかわらず、リアルタイム OS のディスパッチ処理を模倣することができ

る。

【 0 0 1 4 】

第 2 の発明は、第 1 の発明において、前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択し、前記タスク切り替えスレッドに対して処理開始を指示した後に、前記要求を発したタスク処理スレッドを停止させ、

前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行されていたタスク処理スレッドが停止した後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする。

【 0 0 1 5 】

このような第 2 の発明によれば、先に実行されていたタスク処理スレッドが停止した後に、タスク切り替えスレッドによって次に実行すべきタスク処理スレッドが再開される。このため、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【 0 0 1 6 】

第 3 の発明は、第 2 の発明において、前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行されていたタスク処理スレッドが停止したか否かを所定の時間間隔で調べることを特徴とする。

【 0 0 1 7 】

このような第 3 の発明によれば、マルチスレッド OS が提供するスレッドスケジューリングアルゴリズムにかかわらず、先に実行されていたタスク処理スレッドが停止した後に、タスク切り替えスレッドを実行させることができる。

【 0 0 1 8 】

第 4 の発明は、第 1 の発明において、前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択し、前記タスク切り替えスレッドに対して処理開始を指示した後に、前記要求を発したタスク処理スレッドを待機状態にさせ、

前記タスク切り替えスレッドは、処理開始を指示された時には、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドの待機状態を解除するとともに、当該タスク処理スレッドを再開

させることを特徴とする。

【 0 0 1 9 】

このような第 4 の発明によれば、タスク切り替えスレッドが先に実行されていたタスク処理スレッドを停止させた後に、次に実行すべきタスク処理スレッドが再開される。このため、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【 0 0 2 0 】

第 5 の発明は、第 2 または第 4 の発明において、前記タスク切り替え指示手段は、前記タスク切り替えスレッドが処理可能状態となった後に、前記タスク切り替えスレッドに処理開始を指示することを特徴とする。

【 0 0 2 1 】

このような第 5 の発明によれば、複数のタスク処理スレッドが同時にタスク切り替えスレッドに処理開始を指示することがない。このため、タスク切り替えスレッドは、ディスパッチ処理を確実に行うことができる。

【 0 0 2 2 】

第 6 の発明は、第 1 の発明において、前記タスク切り替え指示手段は、次に実行すべきタスク処理スレッドを選択した後に、前記タスク切り替えスレッドに対して処理開始を指示し、

前記タスク切り替えスレッドは、前記タスク処理スレッドよりも高い優先度で実行され、処理開始を指示された時には、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする。

【 0 0 2 3 】

このような第 6 の発明によれば、指定された優先度どおりにスレッドを実行させるマルチスレッド OS を用いた場合に、タスク切り替えスレッドが、タスク処理スレッドよりも優先的に実行される。このため、タスク処理スレッドに待機状態を設けなくても、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【 0 0 2 4 】

第 7 の発明は、第 1 の発明において、前記タスク処理スレッドを生成するタス

ク処理スレッド生成手段をさらに備える。

【 0 0 2 5 】

このような第 7 の発明によれば、タスク処理スレッドをリアルタイム OS シミュレータによって生成し、リアルタイム OS のディスパッチ処理を模倣することができる。

【 0 0 2 6 】

第 8 の発明は、第 1 の発明において、擬似的に割り込みを発生させる割り込みスレッドが発する割り込み要求を受け付け、実行中のタスク処理スレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、次に実行すべきタスク処理スレッドを選択して再開させる割り込み処理手段をさらに備える。

【 0 0 2 7 】

このような第 8 の発明によれば、割り込みスレッドは、割り込み処理手段を用いることにより、実行中のタスク処理スレッドを停止させて、割り込みハンドラを呼び出す。これにより、マルチスレッド OS が提供するスレッドスケジューリングアルゴリズムにかかわらず、リアルタイム OS の割り込み処理と遅延ディスパッチ処理とを模倣することができる。

【 0 0 2 8 】

第 9 の発明は、第 8 の発明において、前記割り込み処理手段は、前記割り込みスレッドから割り込み要求を受け付けた時に他の割り込みスレッドが実行されていた場合には、実行中の割り込みスレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、停止させた割り込みスレッドを再開させることを特徴とする。

【 0 0 2 9 】

このような第 9 の発明によれば、割り込みスレッドは、割り込み処理手段を用いることにより、実行中の割り込みスレッドを停止させて、後の割り込みに対応した割り込みハンドラを先に呼び出す。これにより、リアルタイム OS の多重割り込み処理を模倣することができる。

【 0 0 3 0 】

第 1 0 の発明は、第 8 の発明において、前記割り込みスレッドの中に、所定の時間間隔で擬似的に割り込みを発生させるシステムクロック割り込みスレッドが含まれることを特徴とする。

【 0 0 3 1 】

このような第 1 0 の発明によれば、システムクロック割り込みスレッドによって所定の時間間隔で割り込みを擬似的に発生させることにより、リアルタイム O S のタイマー管理機能を模倣することができる。

【 0 0 3 2 】

第 1 1 の発明は、第 8 の発明において、前記割り込みスレッドを生成する割り込みスレッド生成手段をさらに備える。

【 0 0 3 3 】

このような第 1 1 の発明によれば、割り込みスレッドをリアルタイム O S シミュレータによって生成し、リアルタイム O S の割り込み処理を模倣することができる。

【 0 0 3 4 】

第 1 2 の発明は、リアルタイム O S 上で実行される複数のタスクのそれぞれに汎用のマルチスレッド O S 上で実行されるタスク処理スレッドを割り当て、前記リアルタイム O S の動作を前記マルチスレッド O S 上で模倣するリアルタイム O S のシミュレーション方法をコンピュータで実行させるための記録媒体を記録したコンピュータ読み取り可能な記録媒体であって、

前記リアルタイム O S と同一の条件下で前記タスク処理スレッドが発する要求を受け付け、前記要求に応じてタスクの切り替えを指示する第 1 のステップと、

前記マルチスレッド O S の機能を用いて前記タスク処理スレッドを停止および再開させることにより、前記タスク処理スレッドの中から選択した一のタスク処理スレッドを実行させる第 2 のステップとを備える。

【 0 0 3 5 】

このような第 1 2 の発明によれば、一のタスク処理スレッドのみが切り替えて実行されるので、マルチスレッド O S が提供するスレッドスケジューリングアルゴリズムにかかわらず、リアルタイム O S のディスパッチ処理を模倣することが

できる。

【 0 0 3 6 】

第 1 3 の発明は、第 1 2 の発明において、前記第 1 のステップは、次に実行すべきタスク処理スレッドを選択した後に、前記要求を発したタスク処理スレッドを停止させ、

前記第 2 のステップは、直前に実行されていたタスク処理スレッドが停止した後に、次に実行すべく選択されたタスク処理スレッドを再開させることを特徴とする。

【 0 0 3 7 】

このような第 1 3 の発明によれば、先に実行されていたタスク処理スレッドが停止した後に、第 2 のステップより次に実行すべきタスク処理スレッドが再開される。このため、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【 0 0 3 8 】

第 1 4 の発明は、第 1 2 の発明において、前記第 1 のステップは、次に実行すべきタスク処理スレッドを選択した後に、前記要求を発したタスク処理スレッドを待機状態にさせ、

前記第 2 のステップは、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく選択されたタスク処理スレッドの待機状態を解除するとともに、当該タスク処理スレッドを再開させることを特徴とする。

【 0 0 3 9 】

このような第 1 4 の発明によれば、第 2 のステップにより、先に実行されていたタスク処理スレッドが停止された後に、次に実行すべきタスク処理スレッドが再開される。このため、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【 0 0 4 0 】

第 1 5 の発明は、第 1 2 の発明において、前記第 1 のステップは、次に実行すべきタスク処理スレッドを選択し、

前記第 2 のステップは、前記タスク処理スレッドよりも高い優先度で実行され、直前に実行されていたタスク処理スレッドを停止させた後に、次に実行すべく

選択されたタスク処理スレッドを再開させることを特徴とする。

【 0 0 4 1 】

このような第 1 5 の発明によれば、指定された優先度どおりにスレッドを実行させるマルチスレッド O S を用いた場合に、第 2 のステップが、タスク処理スレッドよりも優先的に実行される。このため、タスク処理スレッドに待機状態を設けなくても、必ず一のタスク処理スレッドのみが実行可能な状態となる。

【 0 0 4 2 】

第 1 6 の発明は、第 1 2 の発明において、前記リアルタイム O S のシミュレーション方法は、擬似的に割り込みを発生させる割り込みスレッドが発する割り込み要求を受け付け、実行中のタスク処理スレッドを停止させ、当該割り込み要求に対応した割り込みハンドラを呼び出した後に、次に実行すべきタスク処理スレッドを選択して再開させる第 3 のステップをさらに備える。

【 0 0 4 3 】

このような第 1 6 の発明によれば、第 3 のステップにより、実行中のタスク処理スレッドが停止され、割り込みハンドラが呼び出される。これにより、マルチスレッド O S が提供するスレッドスケジューリングアルゴリズムにかかわらず、リアルタイム O S の割り込み処理を模倣することができる。

【 0 0 4 4 】

【発明の実施の形態】

(第 1 の実施形態)

図 1 は、本発明の第 1 の実施形態に係るリアルタイム O S シミュレータが動作するコンピュータシステムのハードウェア構成図である。図 1 に示すコンピュータ 1 は、CPU 2、タイマー回路 3、メインメモリ 4、ハードディスク 5、キーボード 6、マウス 7、ディスプレイ 8 および通信ポート 9 を備える。CPU 2 は、ハードディスク 5 に予め記憶されたソフトウェアをメインメモリ 4 に転送して実行する。タイマー回路 3 は、所定の時間間隔で割り込み信号を発生させ、CPU 2 に対して出力する。コンピュータ 1 は、キーボード 6 やマウス 7 から入力される利用者からの指示に従い、ディスプレイ 8 に対して画面表示を行う。また、コンピュータ 1 は、通信ポート 9 を経由して他のコンピュータなどとデータ通信

を行う。コンピュータ 1 には、汎用のマルチスレッド OS が搭載される。汎用 OS 上で実行される各スレッドは、自スレッドまたは他のスレッドを任意のタイミングで停止または再開させることができる。また、この汎用 OS は、イベント機能を有するものとする。イベント機能によれば、各スレッドは、特定のイベントを発生させ、あるいは、特定のイベントが発生するまで待機することができる。

## 【 0 0 4 5 】

図 2 は、本発明の第 1 の実施形態に係るリアルタイム OS シミュレータのソフトウェア構成図である。実システムでは、複数のタスクがリアルタイム OS 上で並行に実行され、複数の割り込みが非同期に発生する。リアルタイム OS シミュレータ 1 0 では、実システムにおける各タスクは、汎用 OS 上で実行されるスレッド（以下、タスク処理スレッドという）にそれぞれ対応づけられる。実システムにおける各割り込みも、同様に汎用 OS 上で実行されるスレッド（以下、割り込みスレッドという）にそれぞれ対応づけられる。

## 【 0 0 4 6 】

図 2 に示したリアルタイム OS シミュレータ 1 0 では、例として、3 つのタスクと 3 つの割り込みとに対応して、第 1 から第 3 のタスク処理スレッド 2 1 ~ 2 3 と、第 1 から第 3 の割り込みスレッド 3 1 ~ 3 3 とが並行に実行される。各タスク処理スレッド 2 1 ~ 2 3 は、それぞれ、実システムにおけるタスクの機能を行うタスク関数を呼び出す。また、タスク処理スレッドは、コンピュータ 1 の記憶装置や入出力機器に対して入出力を行うデバッグ用の入出力関数を呼び出す場合がある。例えば、タスク処理スレッドは、入出力関数を呼び出ることにより、汎用 OS の機能を用いてハードディスク 5 からデータを読み出し、ディスプレイ 8 にデータを表示する。

## 【 0 0 4 7 】

割り込みスレッド 3 1 ~ 3 3 は、実システムにける割り込みを模倣して、擬似的な割り込みを発生させる。また、各割り込みスレッドは、各割り込みに対応した割り込みハンドラを含む。擬似的な割り込みを発生させるタイミングは、コンピュータ 1 の入力機器などにより与えられ、汎用 OS の機能によって各割り込みスレッドに供給される。例えば、タイマー回路 3 によるタイマー割り込み、キー

ボード6やマウス7などにおける利用者の操作、あるいは、通信ポート9におけるデータ送受信などが、擬似的な割り込みを発生させるタイミングとなる。本実施形態では、割り込みスレッド33は、システムクロック割り込みに対応して擬似的な割り込みを発生させるシステムクロック割り込みスレッドであるとする。

#### 【0048】

リアルタイムOSシミュレータ10は、チェンジャースレッド11、システム関数12、割り込み処理関数13、および、システムクロック割り込みスレッド33を備える。システム関数12は、実システムにおいてタスクがリアルタイムOSを呼び出す場合と同じ方法で、各タスク処理スレッド21～23から呼び出される。また、割り込み処理関数13は、割り込みスレッド31～33が擬似的な割り込みを発生させた時に呼び出される。チェンジャースレッド11は、タスク処理スレッド21～23および割り込みスレッド31～33と並行に実行される。チェンジャースレッド11、システム関数12、および、割り込み処理関数13からは、後述するように、汎用OSが提供するスレッド停止／再開機能とイベント機能とが呼び出され、これによりスレッドの実行が制御される。

#### 【0049】

リアルタイムOSシミュレータ10は、タスク関数および割り込みハンドラとともに、メインメモリ4上に転送され、CPU2によって実行される。これにより、実システムにおけるタスクに対応したタスク処理スレッド21～23は、ハードディスク5やディスプレイ8に対してデータの入出力を行うことができる。また、タスク処理スレッドが実行される間に、擬似的な割り込みを発生させることができる。このように、利用者は、図1に示すコンピュータ1上で図2に示すソフトウェアを実行することにより、実システムにおけるアプリケーションソフトウェアに含まれるタスク関数と割り込みハンドラとをデバッグすることができる。

#### 【0050】

以下、リアルタイムOSシミュレータ10を構成するソフトウェアの詳細を説明する。図3は、リアルタイムOSシミュレータ10のメイン処理のフローチャートである。リアルタイムOSシミュレータ10には、実システムにおけるタス

クの機能を行うタスク関数と、実システムにおける割り込みハンドラとが提供される。リアルタイムOSシミュレータ10は、まず、チェンジャースレッド11とシステムクロック割り込みスレッド33とを生成する（ステップS101、S102）。次に、リアルタイムOSシミュレータ10は、与えられたタスク関数のそれぞれに対応して、タスク処理スレッドを順次生成する（ステップS103、S104）。その後、リアルタイムOSシミュレータ10は、割り込みスレッドからの割り込みを許可し（ステップS105）、以降は所定の時間だけスリープする処理を繰り返す（ステップS106）。リアルタイムOSシミュレータ10がステップS106に到達した時点から、図2に示す7つのスレッドが並行に実行される。以降のリアルタイムOSシミュレータ10の機能は、チェンジャースレッド11、システム関数12、および、割り込み処理関数13によって実現される。

#### 【0051】

リアルタイムOSシミュレータ10は、概ね以下のように動作する。タスク処理スレッド21～23は、システム関数12を呼び出した時にタスクの切り替えを行う場合には、次に実行すべきタスク処理スレッドを選択して、チェンジャースレッド11に対してディスパッチ処理開始を指示した後に、自ら停止する。チェンジャースレッド11は、タスク処理スレッドから処理開始を指示されると、次に実行すべく選択されたタスク処理スレッドを再開させる。また、割り込みスレッド31～33は、他のスレッドと並行に実行され、擬似的な割り込みを発生させるタイミングで割り込み処理関数13を呼び出す。割り込み処理関数13を呼び出した割り込みスレッドは、実行中のスレッドを停止させ、自スレッドを実行中のスレッドとして記録する。その後、割り込みスレッドは、対応した割り込みハンドラを呼び出した後に、次に実行すべきスレッドを選択して再開させる。スレッド間の同期を実現するため、2種類のイベントが用いられる。第1のイベントは、タスク処理スレッド21～23がチェンジャースレッド11に対してディスパッチ処理開始を指示するイベントである。第2のイベントは、チェンジャースレッド11と割り込みスレッドとが他のスレッドに対してディスパッチ処理や割り込み処理が開始可能となった旨を通知するイベントである。

## 【0052】

図4は、チェンジャースレッド11の動作を示すフローチャートである。チェンジャースレッド11は、最初に第2のイベントを設定した後（ステップS201）、ステップS202からS207までの処理を繰り返す。チェンジャースレッド11は、まず、ディスパッチ処理開始を指示する第1のイベント待ち状態となり（ステップS202）、第1のイベントが設定された後にステップS203に進む。次に、チェンジャースレッド11は、実行中のタスク処理スレッドが停止するまで、所定の時間だけスリープする処理を繰り返す（ステップS203、S204）。これにより、タスク処理スレッドが実行されている間、チェンジャースレッド11は実行されないことが保証される。実行中のタスク処理スレッドが停止した後、チェンジャースレッド11は、次に実行すべく選択されたタスク処理スレッドを実行スレッドとして記録し（ステップS205）、そのタスク処理スレッドを再開させる（ステップS206）。その後、チェンジャースレッド11は、ディスパッチ処理や割り込み処理が開始可能となった旨を示す第2のイベントを設定し（ステップS207）、ステップS202に戻る。

## 【0053】

図5は、タスク処理スレッドから呼び出されるシステム関数12のフローチャートである。システム関数を呼び出したタスク処理スレッドは、まず、リアルタイムOSが提供する機能のうち、スレッドのディスパッチ以外の処理を行う（ステップS301）。例えば、リアルタイムOSにおける実行タスク切り替え処理では、実行中のタスクがタスク待ち行列の最後尾に移動され、タスク待ち行列の先頭にあるタスクが実行されるようになる。これに対応して、リアルタイムOSシミュレータ10でも、ステップS301において同様の処理が行われる。次に、タスク処理スレッドは、ステップS301における処理の結果、ディスパッチ処理を行うか否かを判断し（ステップS302）、行う場合にはステップS303からS305までの処理を行う。

## 【0054】

タスク処理スレッドは、ディスパッチ処理を行う場合には、まず、第2のイベント待ち状態となり、チェンジャースレッド11がディスパッチ処理を開始可能

となるまで待機する（ステップ S 3 0 3）。チェンジャースレッド 1 1 がディスパッチ処理を開始可能となった後、タスク処理スレッドは、チェンジャースレッド 1 1 に対してディスパッチ処理開始を指示する第 1 のイベントを設定し（ステップ S 3 0 4）、自スレッドを停止させる（ステップ S 3 0 5）。これにより、ステップ S 3 0 5 以降、タスク処理スレッドは停止し、チェンジャースレッド 1 1 が実行される。

## 【 0 0 5 5 】

図 6 ないし図 8 は、本実施形態に係るリアルタイム OS シミュレータによるディスパッチ処理のタイミングチャートの例である。以下に示すタイミングチャートでは、水平方向の直線はそれぞれ 1 つのスレッドを表し、直線の種類はスレッドの状態を表す。太線、細線および破線は、それぞれ、スレッドが実行状態、実行可能状態、および、停止状態または待機状態にあることを示す。また、●印はシステム関数の呼び出しを、黒塗り三角印はイベント設定を、△印はイベント待ちを、□印はスレッドの停止を、黒塗り四角印はスレッドの再開を、◆印は実行スレッド記録処理をそれぞれ表す。

## 【 0 0 5 6 】

図 6 は、実システムにおける第 1 のタスクから第 2 のタスクへの切り替えに対応した、ディスパッチ処理のタイミングチャートの例である。初期状態では、チェンジャースレッドは第 1 のイベント待ち状態にあり（ステップ S 2 0 2）、第 1 のタスク処理スレッドが実行されているとする。第 1 のタスク処理スレッドは、時刻  $t_1$  でシステム関数を呼び出し、時刻  $t_1$  から時刻  $t_2$  の間にディスパッチ以外の処理を行う（ステップ S 3 0 1）。次に、第 1 のタスク処理スレッドは、ディスパッチ処理を行うと判断し（ステップ S 3 0 2）、時刻  $t_2$  で第 2 のイベント待ち状態となる（ステップ S 3 0 3）。この時点で第 2 のイベントは既に設定されているので、第 1 のタスク処理スレッドは、直ちにステップ S 3 0 4 に進み、時刻  $t_3$  で第 1 のイベントを設定する。このため、時刻  $t_3$  以降、チェンジャースレッドは実行可能状態となる。

## 【 0 0 5 7 】

図 6 に示すタイミングチャートでは、第 1 のタスク処理スレッドが引き続き実

行され、時刻  $t_4$  で自スレッドを停止させる（ステップ S 3 0 5）。実行中のタスク処理スレッドが時刻  $t_4$  で停止した後、チェンジャースレッドは、時刻  $t_5$  で第 2 のタスク処理スレッドを実行スレッドとして記録し（ステップ S 2 0 5）、時刻  $t_6$  で第 2 のタスク処理スレッドを再開させる（ステップ S 2 0 6）。このため、時刻  $t_6$  以降、第 2 のタスク処理スレッドは、実行可能状態となる。チェンジャースレッドは、時刻  $t_7$  で第 2 のイベントを設定し、ディスパッチ処理を開始可能となった旨を通知する（ステップ S 2 0 7）。その後、チェンジャースレッド 1 1 は、時刻  $t_8$  で第 1 のイベント待ち状態となる（ステップ S 2 0 2）。このため、時刻  $t_8$  以降、第 2 のタスク処理スレッドが実行される。このようにして、第 1 のタスク処理スレッドから第 2 のタスク処理スレッドへのディスパッチ処理が行われる。ディスパッチに必要な時間は、図 6 において期間 T で示した時刻  $t_2$  から  $t_7$  までの時間である。

## 【 0 0 5 8 】

図 6 に示すタイミングチャートでは、時刻  $t_3$  から時刻  $t_4$  までの期間、および、時刻  $t_6$  から時刻  $t_8$  までの期間で、2 つのスレッドが同時に実行可能状態となる。このような場合にいずれのスレッドが実行されるかは、汎用 OS が提供するスレッドスケジューリングアルゴリズムに依存する。

## 【 0 0 5 9 】

図 7 は、時刻  $t_3$  以降にチェンジャースレッドが実行される場合のタイミングチャートの例である。この場合、チェンジャースレッドは、実行中のタスク処理スレッドが停止するまで、所定の時間だけスリープする処理を繰り返す（ステップ S 2 0 3、S 2 0 4）。チェンジャースレッドは時刻  $t_3 - 1$  でスリープ状態に入り、これ以降、第 1 のタスク処理スレッドが再び実行される。第 1 のタスク処理スレッドは、時刻  $t_4$  で自スレッドを停止させる（ステップ S 3 0 5）。チェンジャースレッドは、時刻  $t_4 - 1$  でスリープ状態から抜け、第 1 のタスク処理スレッドが停止した後に、ステップ S 2 0 5 から S 2 0 7 までの処理を行う。このため、時刻  $t_4 - 1$  以降のタイミングチャートは、図 6 と一致する。このようにチェンジャースレッドは実行中のタスク処理スレッドが停止するまでスリープする処理を繰り返すので、汎用 OS が提供するスレッドスケジューリングアル

ゴリズムにかかわらず、リアルタイムOSのディスパッチ処理を模倣することができる。

## 【 0 0 6 0 】

図8は、時刻 $t_6$ 以降に第2のタスク処理スレッドが実行される場合のタイミングチャートの例である。初期状態から時刻 $t_6$ までのタイミングチャートは、図6と一致する。時刻 $t_6$ で再開された第2のタスク処理スレッドは、時刻 $t_6 - 1$ でシステム関数を呼び出し、時刻 $t_6 - 2$ で第2のイベント待ち状態となる（ステップS303）。このため、時刻 $t_6 - 2$ 以降、チェンジャースレッドが再び実行され、時刻 $t_7$ で第2のイベントを設定する（ステップS207）。このように汎用OSが提供するイベント機能を用いることにより、複数のタスク処理スレッドが同時にディスパッチ処理を開始することがない。このため、チェンジャースレッドは、確実にディスパッチ処理を行うことができる。

## 【 0 0 6 1 】

図9は、割り込みスレッドから呼び出される割り込み処理関数13のフローチャートである。割り込み処理関数を呼び出した割り込みスレッドは、まず、第2のイベント待ち状態となり、チェンジャースレッドが割り込み処理を開始可能となるまで待機する（ステップS401）。チェンジャースレッドが割り込み処理を開始可能となった後、割り込みスレッドは、実行中のスレッドを停止させる（ステップS402）。ステップS402において停止されるスレッドは、タスク処理スレッドまたは他の割り込みスレッドのいずれかである。次に、割り込みスレッドは、自スレッドを実行スレッドとして記録し（ステップS403）、第2のイベントを設定して割り込み処理を開始可能となった旨を通知した後に（ステップS404）、自らの割り込みに対応した割り込みハンドラを呼び出す（ステップS405）。

## 【 0 0 6 2 】

割り込みハンドラの処理が終了した後、割り込みスレッドは、多重割り込み処理と遅延ディスパッチ処理とを行うか否かを調べる（ステップS406、S407）。多重割り込み処理を行わずに遅延ディスパッチ処理を行う場合には、割り込みスレッドは、次に実行すべきタスク処理スレッドを実行スレッドとして記録

し（ステップS408）、そのスレッドを再開させる（ステップS409）。それ以外の場合には、割り込みスレッドは、ステップS402において停止させたスレッドを実行スレッドとして記録し（ステップS410）、そのスレッドを再開させる（ステップS411）。

【0063】

図10ないし図13は、本実施形態に係るリアルタイムOSシミュレータによる割り込み処理のタイミングチャートの例である。これらの図は、図6と同じ記法を用いて記述されている。

【0064】

図10は、第1のタスク処理スレッドが実行されている間に第1の割り込みが発生した場合のタイミングチャートである。初期状態では、第2のイベントは設定され、第1のタスク処理スレッドが実行されているとする。第1の割り込みスレッドは、初期状態では実行可能状態にあり、時刻 $t_1$ で割り込み処理関数を呼び出すとする。時刻 $t_1$ 以降、第1の割り込みスレッドは、図9に示したフローチャートに従って動作する。第1の割り込みスレッドは、まず、時刻 $t_2$ で第2のイベント待ち状態となる（ステップS401）。この時点で第2のイベントは既に設定されているので、第1の割り込みスレッドは、直ちにステップS402に進み、時刻 $t_3$ で直前に実行されていたスレッド、すなわち、第1のタスク処理スレッドを停止させる（ステップS402）。このため、時刻 $t_3$ 以降、第1のタスク処理スレッドは、停止状態となる。

【0065】

次に、第1の割り込みスレッドは、時刻 $t_4$ で自スレッドを実行スレッドとして記録し（ステップS403）、時刻 $t_5$ で第2のイベントを設定する（ステップS404）。このため、時刻 $t_5$ 以降、他の割り込み処理が開始可能となる。さらに、第1の割り込みスレッドは、時刻 $t_6$ から時刻 $t_7$ までの期間で自らの割り込みに対応した割り込みハンドラを呼び出す（ステップS405）。その後、第1の割り込みスレッドは、多重割り込み処理および遅延ディスパッチ処理を行わないと判断し（ステップS406、S407）、ステップS410に進む。最後に、第1の割り込みスレッドは、時刻 $t_8$ でステップS402において停止

させた第1のタスク処理スレッドを実行スレッドとして記録し（ステップS410）、時刻 $t_9$ でそのスレッドを再開させる（ステップS411）。これにより、第1のタスク処理スレッドが、再び実行される。このように割り込み処理関数を呼び出した割り込みスレッドが実行中のタスク処理スレッドを停止させ、割り込みハンドラを呼び出した後に、停止させたタスク処理スレッドを再開させることにより、リアルタイムOSの割り込み処理を模倣することができる。

## 【0066】

図11は、遅延ディスパッチ処理のタイミングチャートの例である。初期状態から時刻 $t_7$ までのタイミングチャートは、図10と一致する。遅延ディスパッチ処理を行う場合、第1の割り込みスレッドは、ステップS407からステップS408へ進む。その後、第1の割り込みスレッドは、時刻 $t_8$ で実行スレッドとして第2のタスク処理スレッドを記録し（ステップS408）、時刻 $t_9$ においてそのスレッドを再開させる（ステップS409）。これにより、時刻 $t_9$ 以降、第1のタスク処理スレッドに代えて、第2のタスク処理スレッドが実行される。このように停止させた割り込みスレッド以外の割り込みスレッドを再開させることにより、リアルタイムOSの遅延ディスパッチ処理を模倣することができる。

## 【0067】

図12は、多重割り込み処理のタイミングチャートの例である。なお、図12(b)は、図12(a)に続く時間帯を示したタイミングチャートである。初期状態から時刻 $t_{6-1}$ までのタイミングチャートは、図10と一致する。第1の割り込みスレッドが時刻 $t_5$ で割り込み処理を開始可能となった旨を通知した後、第2の割り込みスレッドが時刻 $t_{6-1}$ で割り込み処理関数を呼び出すとする。第2の割り込みスレッドは、時刻 $t_{6-3}$ で直前に実行されていたスレッド、すなわち、第1の割り込みスレッドを停止させ（ステップS402）、時刻 $t_{6-3}$ から $t_{6-7}$ までの期間にステップS403からS405までの処理を行う。その後、第2の割り込みスレッドは、多重割り込み処理中であると判断して（ステップS406）、ステップS406からステップS410に進む。最後に、第2の割り込みスレッドは、時刻 $t_{6-8}$ で実行スレッドとして第1の割り込み

スレッドを記録し（ステップ S410）し、時刻  $t_{6-9}$  でそのスレッドを再開させる（ステップ S411）。時刻  $t_{6-9}$  以降、再開された第 1 の割り込みスレッドは、第 2 の割り込みスレッドによる割り込みが発生しなかった場合と同様に動作する。このように割り込みスレッド実行中に他の割り込みが発生した時に、先の割り込みスレッドを停止させて、後で発生した割り込みに対応した割り込みハンドラを先に呼び出すことにより、リアルタイム OS の多重割り込み処理を模倣することができる。

## 【0068】

図 13 は、ディスパッチ処理中に割り込みが発生した場合の割り込み処理のタイミングチャートの例である。このタイミングチャートは、図 6 における時刻  $t_4$  で、第 1 の割り込みスレッドが割り込み処理関数を呼び出した場合のものである。第 1 の割り込みスレッドは、時刻  $t_{4-1}$  で第 2 のイベント待ち状態となる（ステップ S401）。このため、時刻  $t_{4-2}$  以降、チェンジャースレッドが実行される。第 1 の割り込みスレッドは、時刻  $t_7$  でチェンジャースレッドが第 2 のイベントを設定した後に実行可能状態となり、時刻  $t_8$  でチェンジャースレッドが第 1 のイベント待ち状態となった後に実行される。このようにディスパッチ処理中に割り込みが発生しても、第 2 のイベントを用いた排他制御を行うことにより、リアルタイム OS におけるディスパッチ処理中の割り込み処理を模倣することができる。

## 【0069】

システムクロック割り込みスレッド 33 は、システムクロックに対応して擬似的に割り込みを発生させ、システムクロック割り込みに対する割り込みハンドラを備える。このような割り込みスレッドを備えることにより、実システムにおけるリアルタイム OS のタイマー管理機能を模倣することができる。

## 【0070】

以上に示すように、本実施形態に係るリアルタイム OS シミュレータでは、タスク処理スレッドは、システム関数を呼び出した時に、次に実行すべきタスク処理スレッドを選択し、チェンジャータスクに対してディスパッチ処理開始を指示した後に、自ら停止する。一方、チェンジャースレッドは、処理開始を指示され

た時には、直前に実行されていたタスク処理スレッドが停止した後に、次に処理すべく選択されたタスク処理スレッドを再開させる。これにより、汎用OSが提供するスレッドスケジューリングアルゴリズムにかかわらず、リアルタイムOSのディスパッチ処理を模倣することができる。また、タスク処理スレッドは、チェンジャースレッドがディスパッチ処理を可能状態になった後に処理開始を指示するので、複数のタスク処理スレッドが同時にディスパッチ処理を行うことがない。さらに、割り込みスレッドは、割り込み処理関数を呼び出した時に、実行中のスレッドを停止させて、対応した割り込みハンドラを呼び出した後に、次に実行すべきタスク処理スレッドを再開させる。これにより、リアルタイムOSの割り込み処理と遅延ディスパッチ処理とを模倣することができる。また、割り込みスレッドは、割り込みスレッド実行中に割り込みが発生した時にも、実行中の割り込みスレッドを停止させ、後の割り込みに対応した割り込みハンドラを呼び出した後に、停止させた割り込みスレッドを再開させる。これにより、リアルタイムOSにおける多重割り込み処理を模倣することができる。このように、本実施形態に係るリアルタイムOSシミュレータは、リアルタイムOS上で実行されるマルチタスクのソフトウェアを開発するために必要とされるリアルタイムOSの機能を模倣することができる。

## 【 0 0 7 1 】

## (第2の実施形態)

本発明の第2の実施形態に係るリアルタイムOSシミュレータは、第1の実施形態に係るリアルタイムOSシミュレータと同一のソフトウェア構成を有し、チェンジャースレッドおよびシステム関数の一部のみが異なる。本実施形態に係るリアルタイムOSシミュレータは、自スレッドを停止させることを禁止する汎用OS上でも動作する。このリアルタイムOSシミュレータでは、各タスク処理スレッドごとに設けられ、各タスク処理スレッドが待機状態にあることを示す第3のイベントが使用される。

## 【 0 0 7 2 】

図14は、本実施形態に係るチェンジャースレッド11の動作を示すフローチャートである。第1の実施形態に係るチェンジャースレッドは、図4に示すよう

に、実行中のタスク処理スレッドが停止するまで、所定の時間だけスリープする処理を繰り返す（ステップ S 2 0 3、S 2 0 4）。これに対し、本実施形態に係るチェンジャースレッドは、実行中のタスク処理スレッドを停止させる（ステップ S 5 0 3）ことを特徴とする。

## 【 0 0 7 3 】

図 1 4 に示すフローチャートにおいて、ステップ S 5 0 3 から S 5 0 5 まで以外の処理は図 4 と同一であるので、説明を省略する。チェンジャースレッドは、ディスパッチ処理開始を指示する第 1 のイベントが設定された（ステップ S 5 0 2）後に、実行中のタスク処理スレッドを停止させる（ステップ S 5 0 3）。次に、チェンジャースレッドは、次に実行すべきタスク処理スレッドが割り込み時の遅延ディスパッチ処理によって停止しているか否かを判断し（ステップ S 5 0 4）、遅延ディスパッチによって停止していると判断した場合には、次に実行すべきタスク処理スレッドに係る第 3 のイベントを設定する（ステップ S 5 0 5）。

## 【 0 0 7 4 】

図 1 5 は、タスク処理スレッドから呼び出されるシステム関数 1 2 のフローチャートである。このフローチャートにおけるステップ S 6 0 1 から S 6 0 4 の処理は、図 5 に示すフローチャートにおけるステップ S 4 0 1 から S 4 0 4 と同一であるので、説明を省略する。システム関数を呼び出したタスク処理スレッドは、ステップ S 6 0 5 において、自タスク処理スレッドに係る第 3 のイベント待ち状態となる。

## 【 0 0 7 5 】

このように、タスク処理スレッドはチェンジャースレッドに対して処理開始を指示した後に第 3 のイベント待ち状態となり、チェンジャースレッドは処理開始を指示されると、実行中のタスク処理スレッドを停止させる。このため、自スレッドを停止させることを禁止する汎用のマルチスレッド OS を用いた場合でも、リアルタイム OS のディスパッチ処理を模倣することができる。

## 【 0 0 7 6 】

図 1 6 および図 1 7 は、本実施形態に係るリアルタイム OS シミュレータによ

るディスパッチ処理のタイミングチャートの例である。図 1 6 および図 1 7 は、いずれも図 6 と同じ記法を用いて記述され、それぞれ図 6 および図 7 に対応する。初期状態では、第 2 のイベントは設定されているとする。また、チェンジャースレッドは第 1 のイベント待ち状態（ステップ S 5 0 2）にあり、第 2 のタスク処理スレッドはシステム関数の中で第 3 のイベント待ち状態（ステップ S 6 0 5）にあるとする。

## 【 0 0 7 7 】

図 1 6 に示すタイミングチャートでは、時刻  $t_1$  でシステム関数を呼び出した第 1 のタスク処理スレッドは、時刻  $t_3$  で第 1 のイベントを設定（ステップ S 6 0 4）した後、時刻  $t_3 - 1$  で第 3 のイベント待ち状態となる（ステップ S 6 0 5）。このため、時刻  $t_3 - 1$  以降、チェンジャースレッドが実行される。チェンジャースレッドは、時刻  $t_4$  で第 1 のタスク処理スレッドを停止させる（ステップ S 5 0 3）。チェンジャースレッドは、次に実行すべきタスク処理スレッドが遅延ディスパッチ処理によって停止しているか否かを判断する（ステップ S 5 0 4）。この例では、チェンジャースレッドは、ステップ S 5 0 4 からステップ S 5 0 5 へ進み、時刻  $t_4 - 1$  で第 2 のタスク処理スレッドに係る第 3 のイベントを設定する（ステップ S 5 0 5）。このため、時刻  $t_4 - 1$  以降、第 2 のタスク処理スレッドは、第 3 のイベント待ち状態から抜けて停止する。その後、チェンジャースレッドは、時刻  $t_5$  で次に実行すべきスレッドとして第 2 のタスク処理スレッドを記録した後（ステップ S 5 0 6）、時刻  $t_6$  で第 2 のタスク処理スレッドを再開させる（ステップ S 5 0 7）。このため、時刻  $t_6$  以降、第 2 のタスク処理スレッドは、実行可能状態となる。さらに、チェンジャースレッドは、時刻  $t_7$  で第 2 のイベントを設定した後（ステップ S 5 0 8）、第 1 のイベント待ち状態となる（ステップ S 5 0 2）。このため、時刻  $t_8$  以降、第 2 のタスク処理スレッドが実行される。

## 【 0 0 7 8 】

図 1 7 は、時刻  $t_3$  以降にチェンジャースレッドが実行される場合のタイミングチャートの例である。このタイミングチャートは、チェンジャータスクが時刻  $t_4$  で第 1 のタスク処理スレッドを停止させる（ステップ S 5 0 3）点でのみ、

図16と異なる。

【0079】

なお、図16および図17のタイミングチャートにおいて、第2のタスク処理スレッドが第3のイベント待ち状態でない場合には、チェンジャースレッドは、ステップS504からステップS506に進み、第2のタスク処理スレッドに係る第3のイベントを設定しない。

【0080】

以上に示すように、本実施形態に係るリアルタイムOSシミュレータでは、タスク処理スレッドは、チェンジャースレッドに対して処理開始を指示した後に待機状態となる。チェンジャースレッドは、処理開始を指示されると実行中のタスク処理スレッドを停止させた後に、次に実行すべきタスク処理スレッドが待機状態にある場合は、その待機状態を解除して再開させる。このため、自スレッドを停止させることを禁止する汎用のマルチスレッドOSを用いた場合でも、リアルタイムOSのディスパッチ処理を模倣することができる。

【0081】

なお、指定された優先度に従ってスレッドを実行させる汎用のマルチスレッドOSを用いる場合には、スケジューラスレッドの優先度をタスク処理スレッドよりも高く設定する方法を用いてもよい。この方法によれば、図14および図15に示すフローチャートから第3のイベントに関する処理を削除しても、スケジューラスレッドがタスク処理スレッドよりも優先的に実行され、本実施形態の効果が損なわれることがない。

【0082】

以上に示した第1および第2の実施形態に係るリアルタイムOSシミュレータについては、リアルタイムOSおよび汎用OSは、特定のOSに限られるものではない。また、リアルタイムOSシミュレータは、汎用OSが提供するイベント機能以外の機能を用いて、スレッド間の同期を実現するものであってもよい。また、タスク処理スレッドおよび割り込みスレッドは、いずれも、リアルタイムOSシミュレータにより生成されても、リアルタイムOSシミュレータの外部で生成されてもよい。

【図面の簡単な説明】

【図 1】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータが動作するコンピュータシステムのハードウェア構成図である。

【図 2】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータのソフトウェア構成図である。

【図 3】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータのメイン処理のフローチャートである。

【図 4】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータにおけるチェンジャースレッドのフローチャートである。

【図 5】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータにおけるシステム関数のフローチャートである。

【図 6】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータによるディスパッチ処理のタイミングチャートの例である。

【図 7】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータによるディスパッチ処理のタイミングチャートの例である。

【図 8】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータによるディスパッチ処理のタイミングチャートの例である。

【図 9】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータの割り込み処理関数のフローチャートである。

【図 1 0】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータによる割り込み処理のタイミングチャートの例である。

【図 1 1】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータによる割り込み処理のタイミングチャートの例である。

【図 1 2】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータによる割り込み処理のタイミングチャートの例である。

【図 1 3】

本発明の第 1 の実施形態に係るリアルタイム OS シミュレータによる割り込み処理のタイミングチャートの例である。

【図 1 4】

本発明の第 2 の実施形態に係るリアルタイム OS シミュレータにおけるチェンジャースレッドのフローチャートである。

【図 1 5】

本発明の第 2 の実施形態に係るリアルタイム OS シミュレータにおけるシステム関数のフローチャートである。

【図 1 6】

本発明の第 2 の実施形態に係るリアルタイム OS シミュレータによるディスパッチ処理のタイミングチャートの例である。

【図 1 7】

本発明の第 2 の実施形態に係るリアルタイム OS シミュレータによるディスパッチ処理のタイミングチャートの例である。

【図 1 8】

従来の制御ソフトウェア実行システムのソフトウェア構成図である。

【符号の説明】

1 0 …リアルタイム OS シミュレータ

1 1 …チェンジャースレッド

1 2 …システム関数

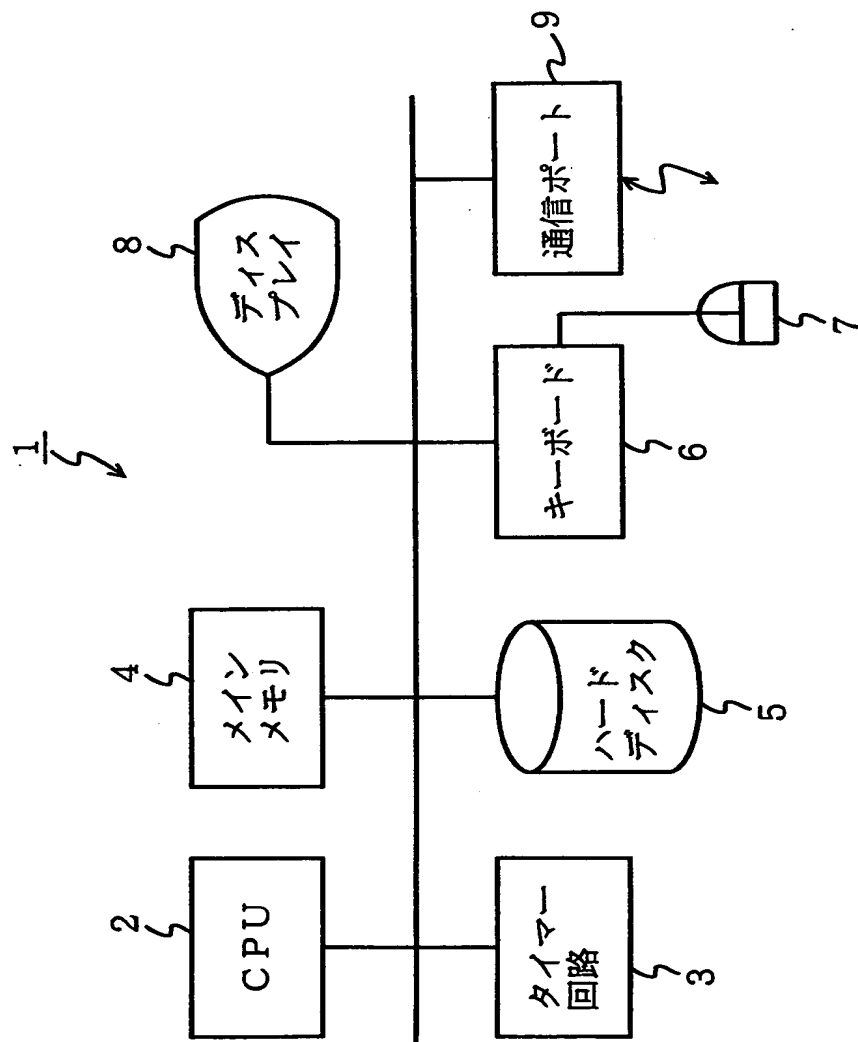
1 3 … 割り込み処理関数

2 1 ～ 2 3 … タスク処理スレッド

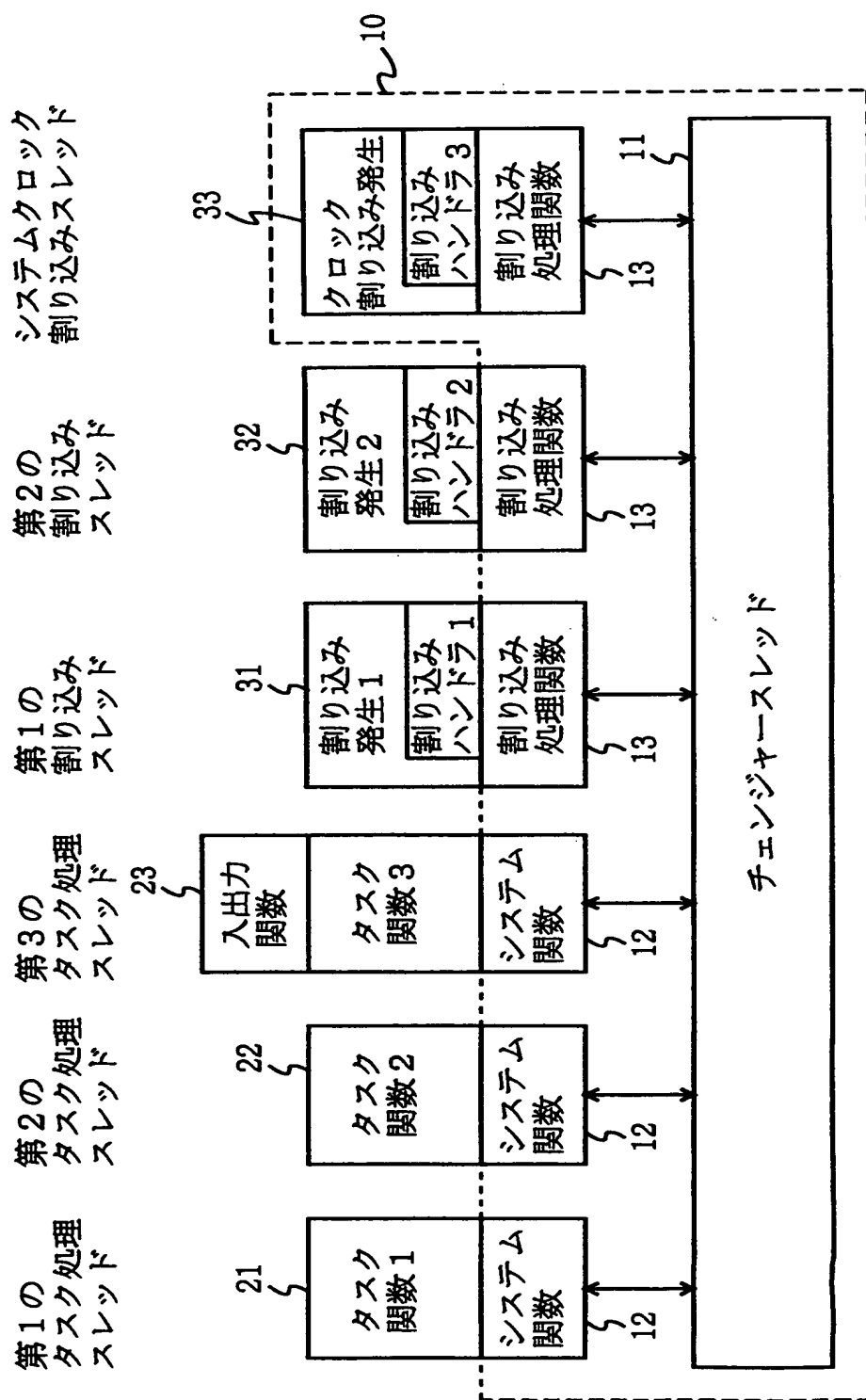
3 1 ～ 3 3 … 割り込みスレッド

【書類名】 図面

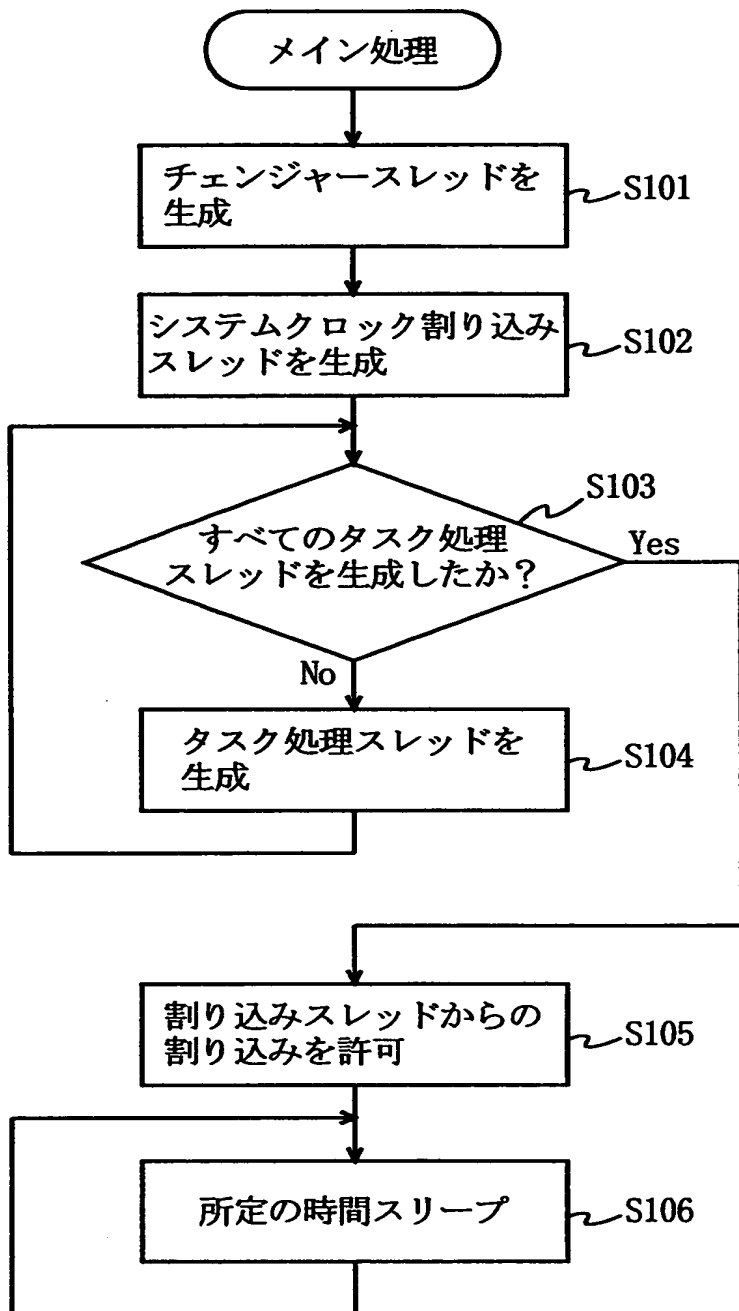
【図 1】



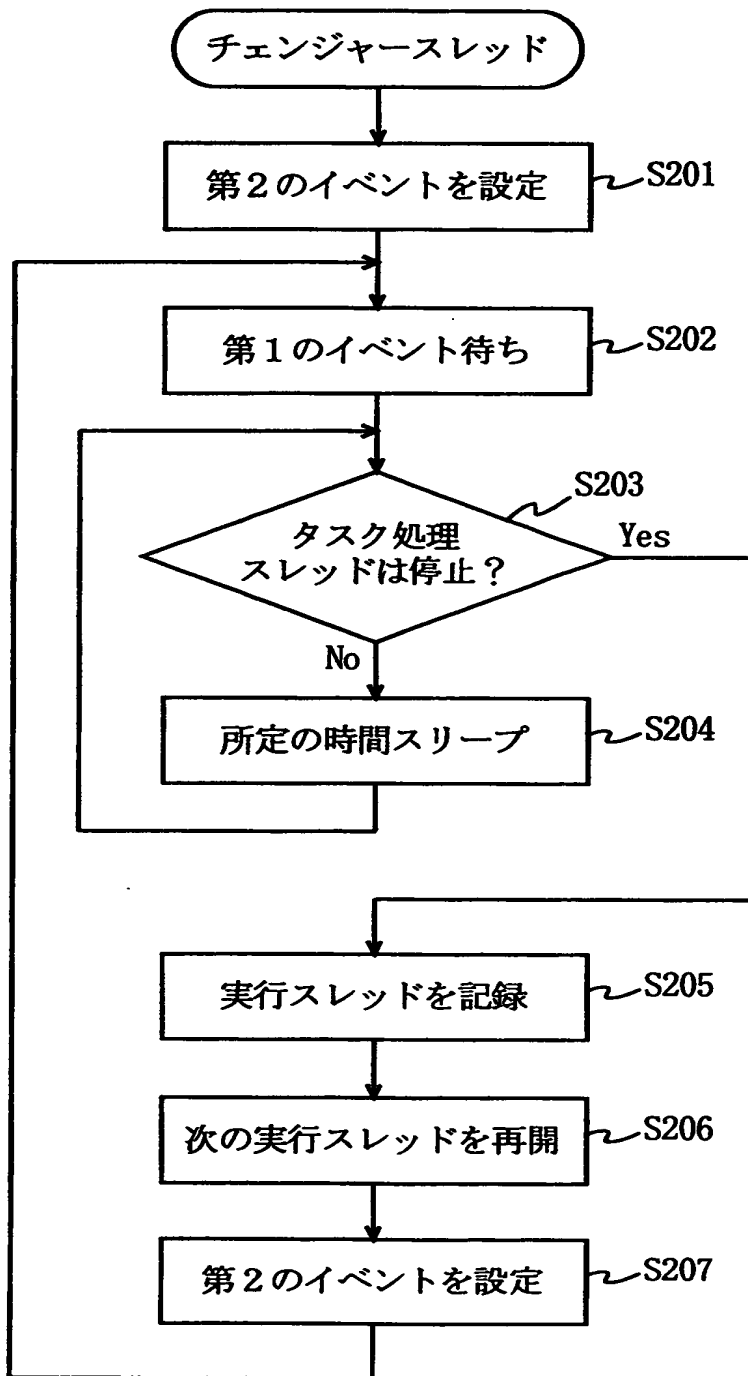
【図 2】



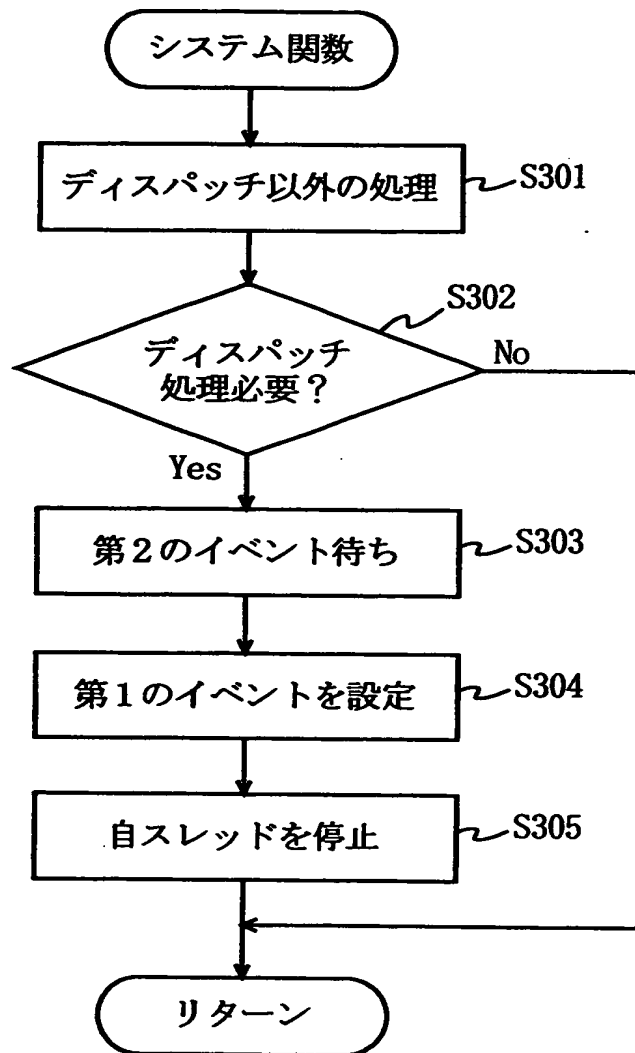
【図3】



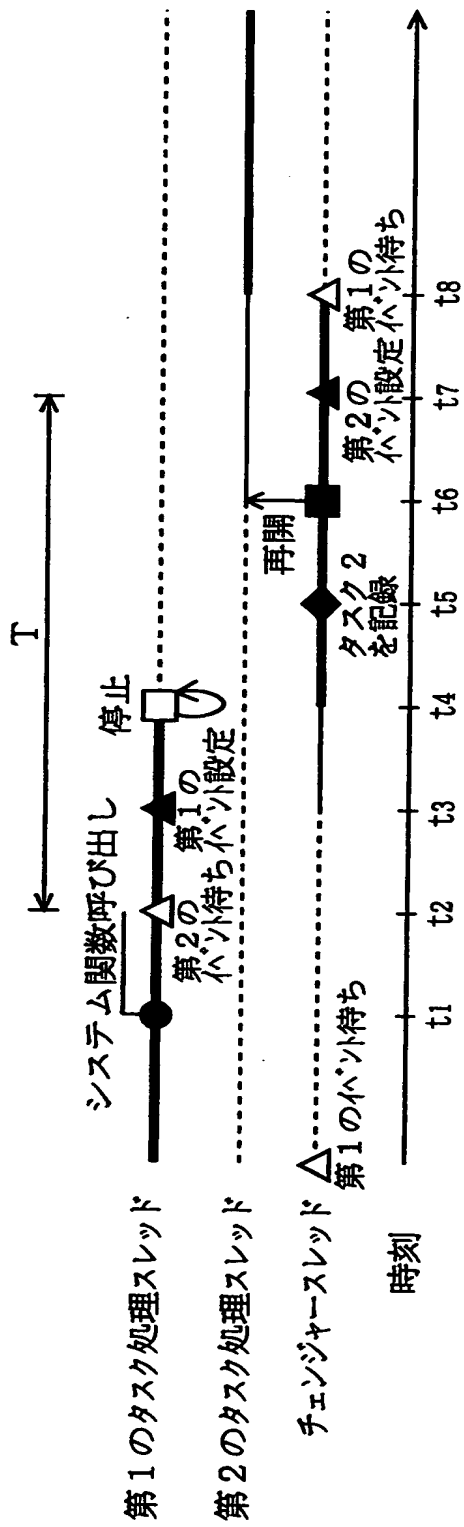
【図 4】



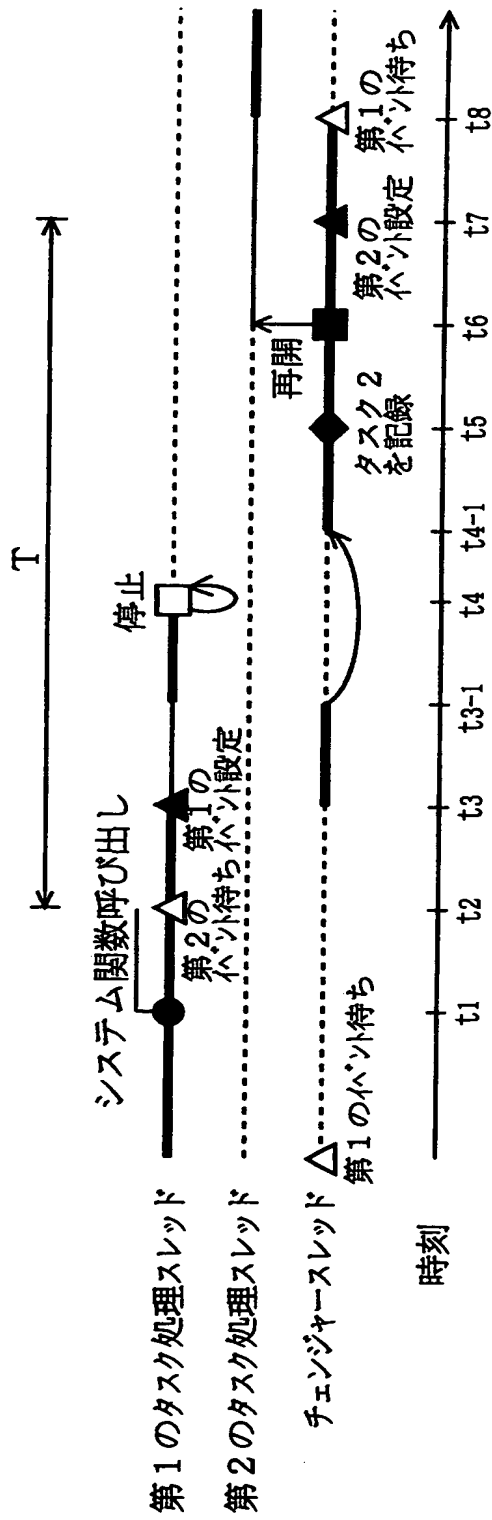
【図 5】



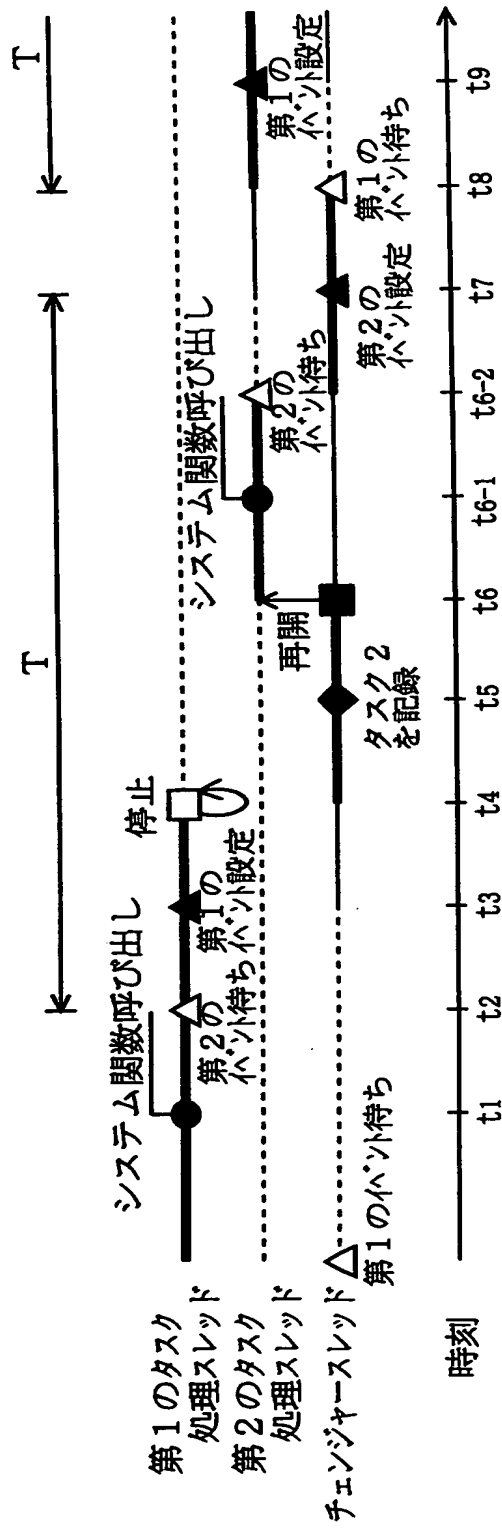
【図 6】



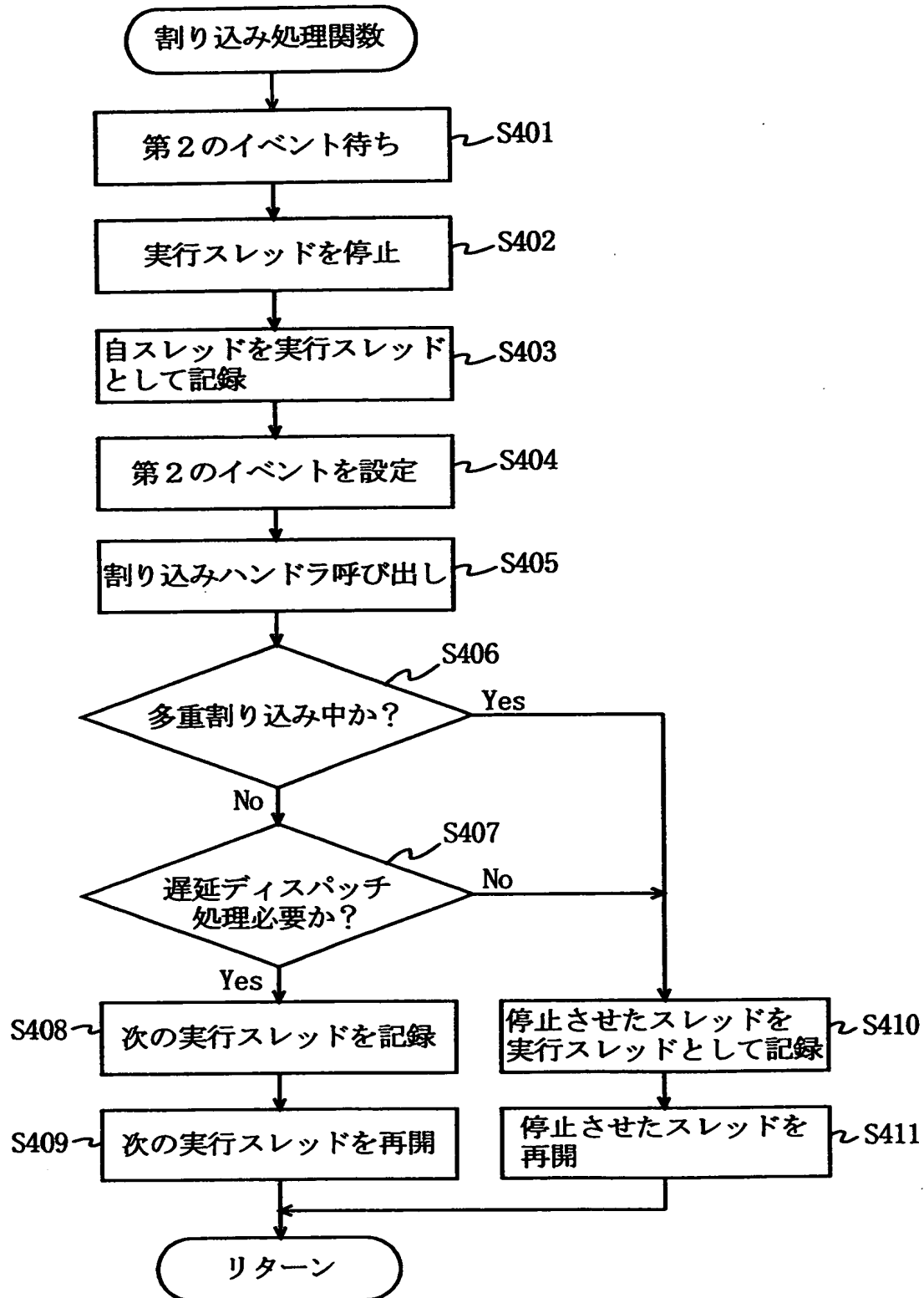
【図 7】



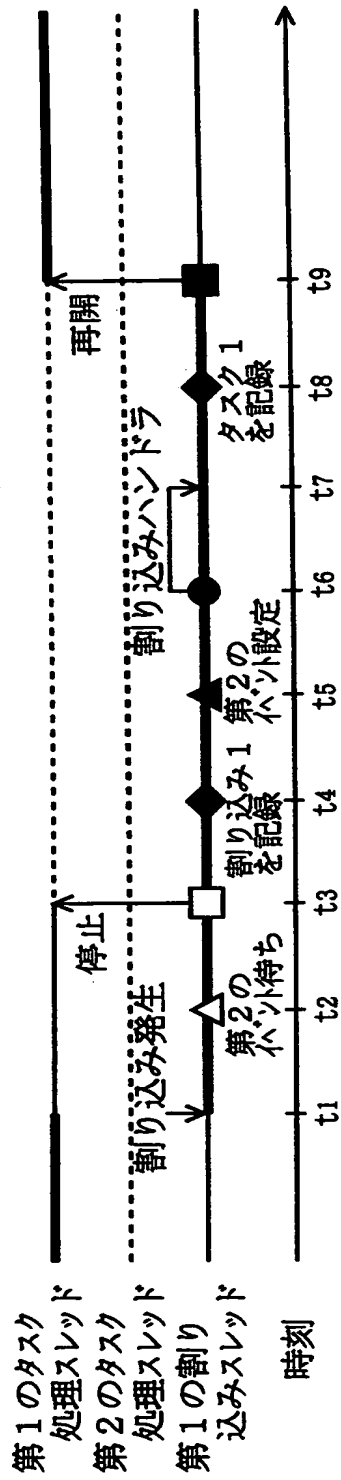
【図 8】



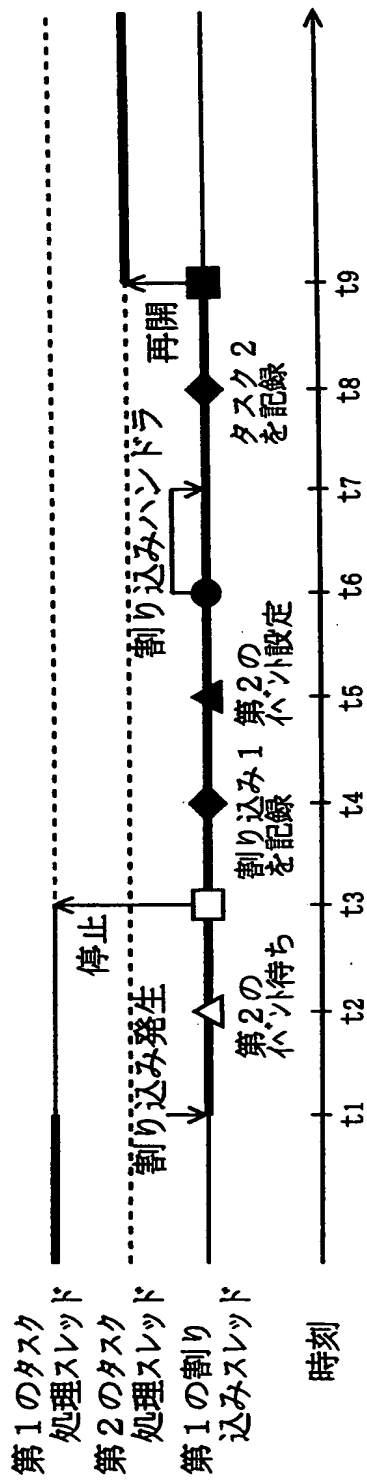
【図 9】



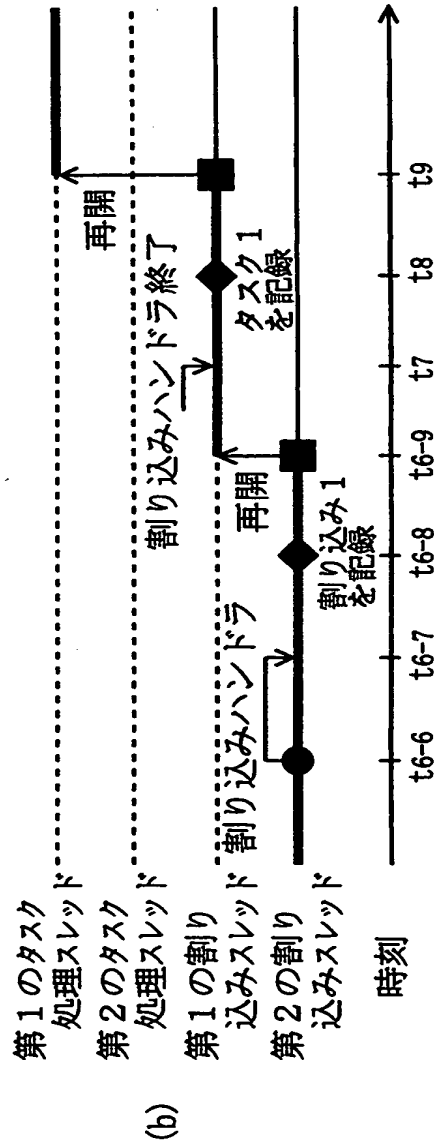
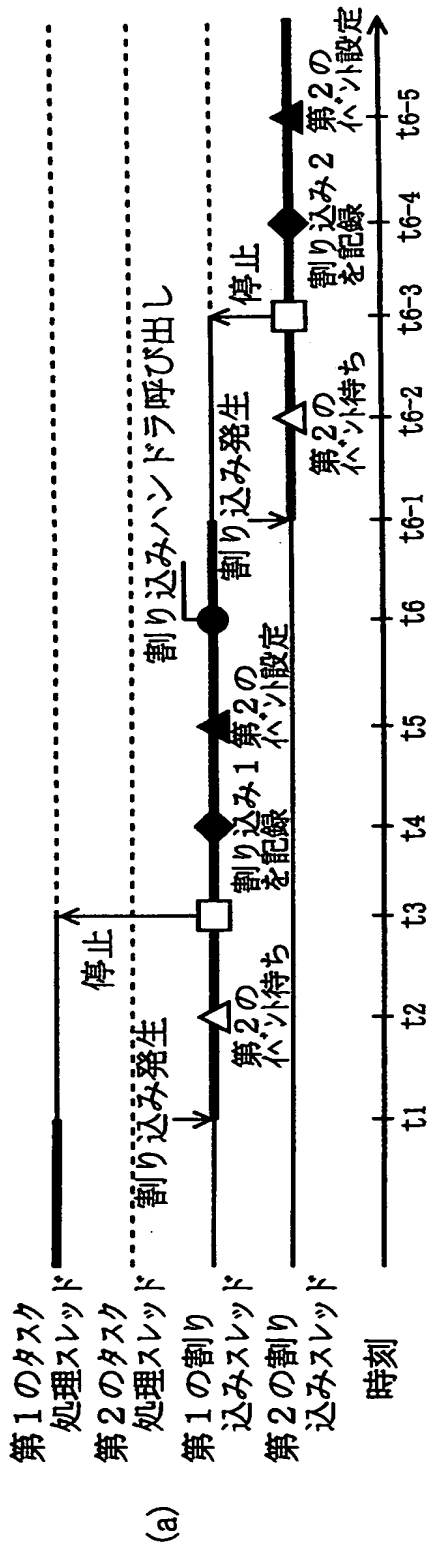
【図 1 0】



【図 1 1】

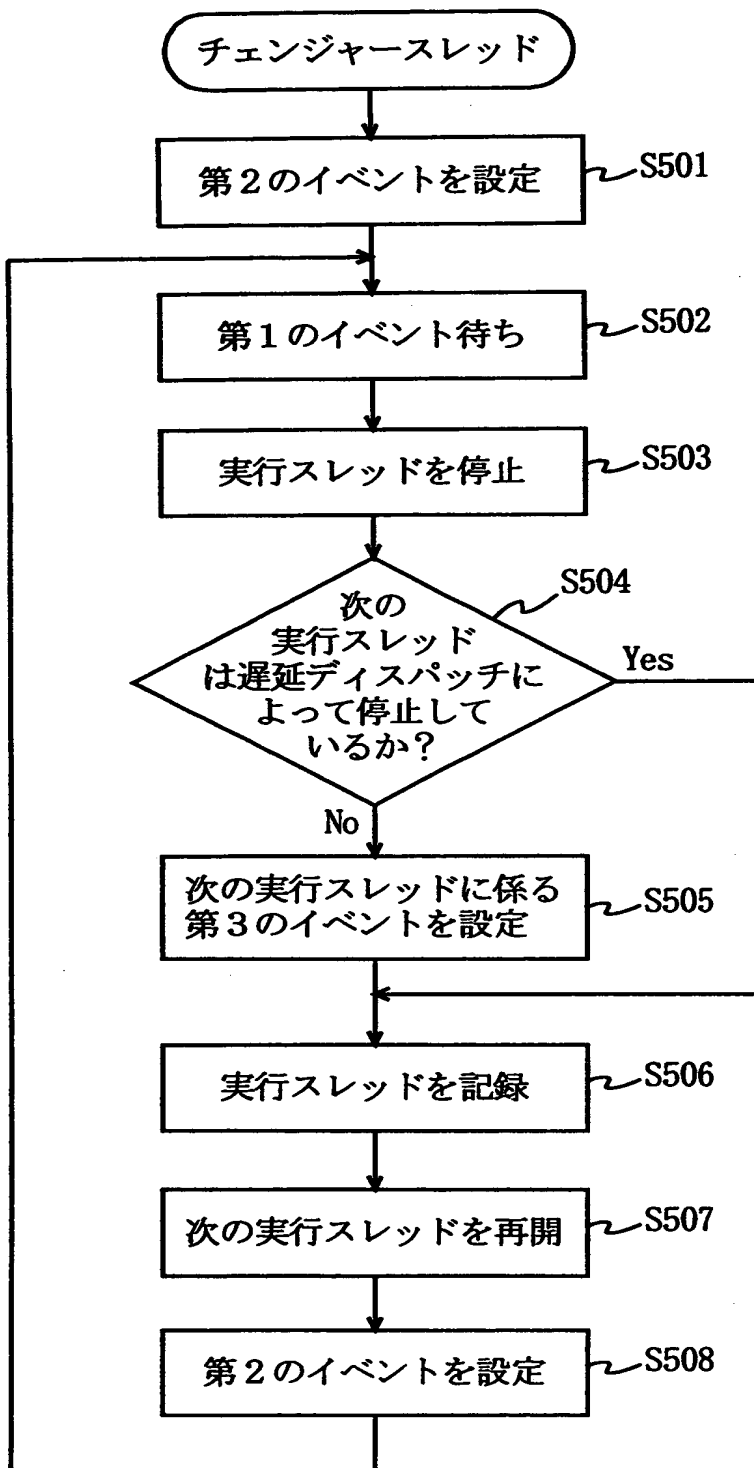


【図 1 2】

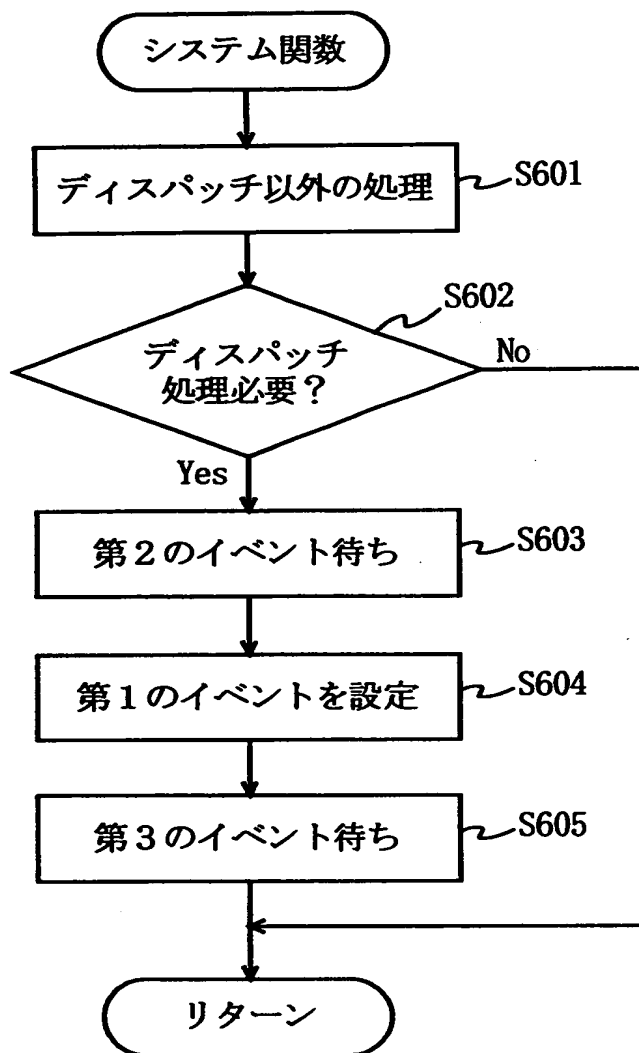




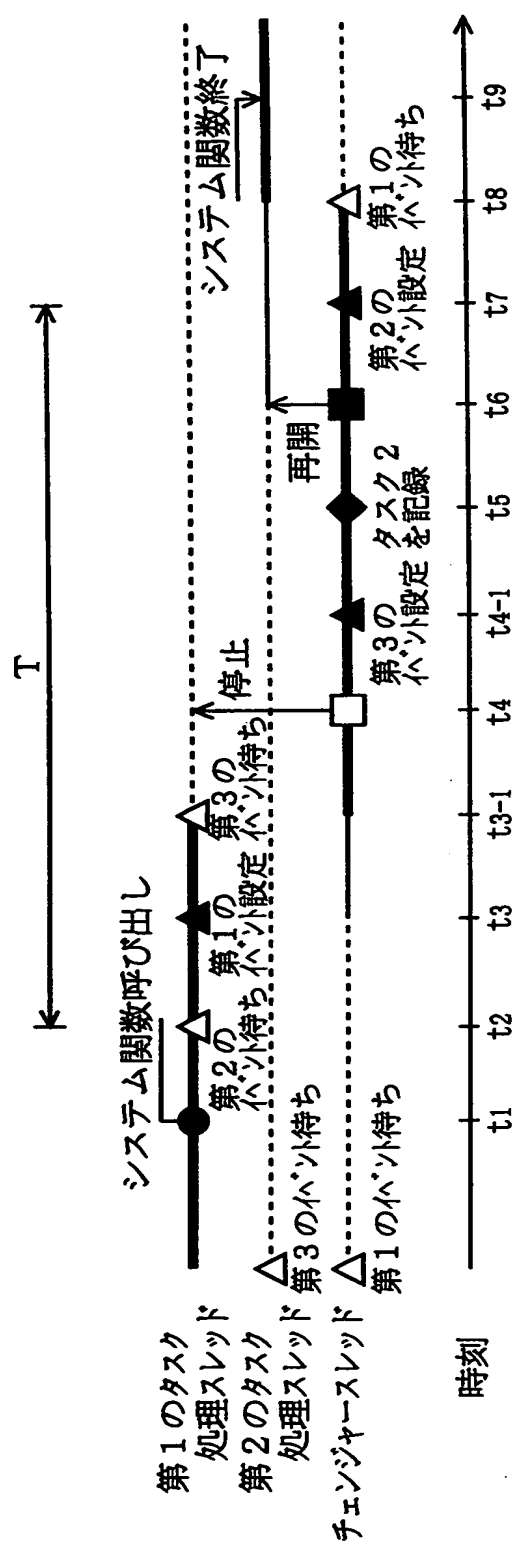
【図 14】



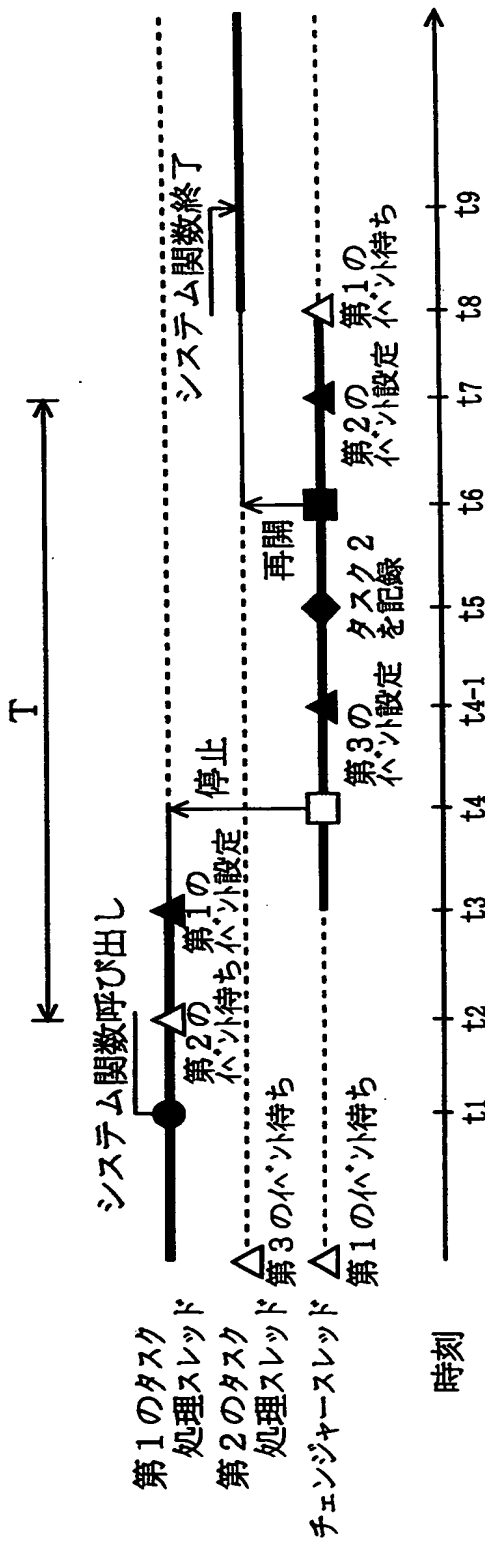
【図 1 5】



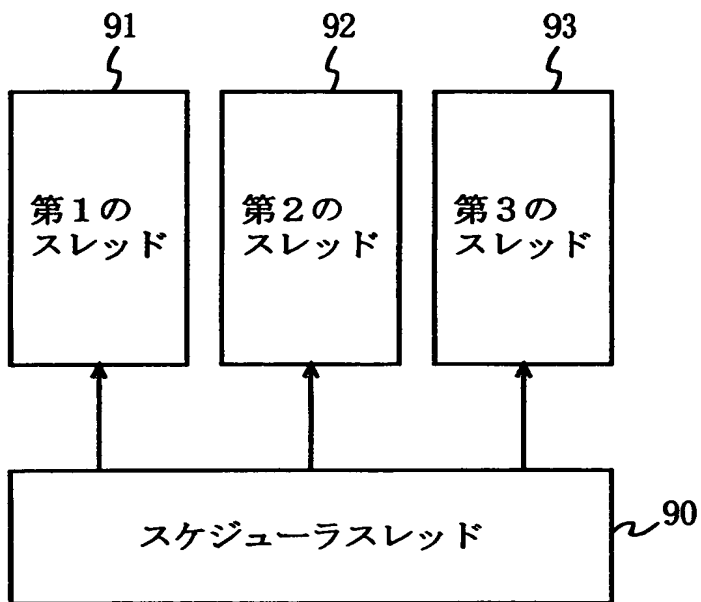
【図16】



【図 17】



【図 1 8】



【書類名】            要約書

【要約】

【課題】    リアルタイムOSのディスパッチ処理と割り込み処理とを、汎用のマルチスレッドOS上で模倣する。

【解決手段】    タスクに対応したスレッド21～23、擬似的な割り込みを発生させるスレッド31～33、スレッドの実行を制御するスレッド11が、マルチスレッドOS上で並行に実行される。システム関数12を呼び出したスレッド21～23は、次に実行すべきスレッドを選択して、スレッド11に処理開始を指示した後、自ら停止する。指示を受けたスレッド11は、実行中のスレッドが停止した後に、選択されたスレッドを再開させる。割り込み処理関数13を呼び出したスレッド31～33は、実行中のスレッドを停止させ、割り込みハンドラを呼び出し後に、次に実行すべきスレッドを選択して再開させる。

【選択図】            図2

特2000-120903

認定・付加情報

特許出願の番号	特願2000-120903
受付番号	50000507667
書類名	特許願
担当官	第七担当上席 0096
作成日	平成12年 4月24日

<認定情報・付加情報>

【提出日】	平成12年 4月21日
-------	-------------

次頁無

特2000-120903

出 願 人 履 歴 情 報

識別番号 [000005821]

1. 変更年月日	1990年 8月28日
[変更理由]	新規登録
住 所	大阪府門真市大字門真1006番地
氏 名	松下電器産業株式会社